REST

Jan Ruthardt

IAAS

Abstract. Diese Ausarbeitung entstand im Rahmen des Seminars Cloud-Computing an der Universität Stuttgart. Die Ausarbeitung behandelt das Thema REST (Representational State Transfer) welches einen Architekturstiel für Webservices darstellt

Keywords: REST;IAAS;Cloud-Computing

1 Introduction

Diese Ausarbeitung entstand im Rahmen des Seminars Cloud-Computing an der Universität Stuttgart. Die Ausarbeitung behandelt das Thema REST (Representational State Transfer) welches einen Architekturstiel für Webservices darstellt. Inhaltlich teilt sich die Ausarbeitung in Zwei große Themenbereiche auf. Der erste Teil beschäftigt sich damit warum REST verwendet wird und welche Eigenschaften zu einer REST Architektur benötigt werden. Dabei wird sowohl auf notwendig, als auch auf optionale Bedingungen eingegangen. Der zweite Teil der Ausarbeitung beschäftigt sich mit den Technologien die für REST benötigt werden. Hier wird insbesondere auf HTTP (Hypertext Transfer Protocol) eingegangen. Es werden alle wichtige Methoden erklärt und auf Authentifizierung und Caching in HTTP eingegangenn. Darüber hinaus werden auch noch verschiedene Daten Repräsentations Fromate vorgestellt die sich für REST eigenen und oft verwendet werden.

2 Warum REST

REST wurde als Architekturstil für das HTTP Protokoll entwickelt. Es baut auf der Version 1.0 auf welche im Jahr 1991 veröffentlicht wurde. REST hingegen wurde erstmals unter dem Namen HTTP Object Modell 1994 von Roy Fielding veröffentlicht. Wenn alle der Bedingungen die im Architekturstil gefordert sind, erfüllt werden zeichnen sich die resultierenden Systeme durch folgende Eigenschaften aus [3]:

1. Lose Kopplung (Verwaltung von getrennten Systemen die über Interfaces Kommunizieren)
2. Interoperabilität (Kommunikation von zwei technisch unterschiedlich Systemen) Dabei unterstützt z.B. fast jedes System die Http-Methoden
3. Performance und Skalierbarkeit (Durch die Lose Kooplung und die Verwendung von HTTP steigt die Skalierbarkeit und Performance enorm)

Dabei setzt REST bei der Repräsentation und Speicherung von Daten auf sogenannte Ressourcen die durch den Uniform Resource Identifier (URI, Der Uniform Resource Identifier ist eine Zeichenkette die zur Adressierung von Ressourcen dient. Sie besteht aus einem Schema, einem Authority und einem Path) adressiert werden. REST ist allerdings Technologie unabhängig designt. Vor allem in der Art wie Daten in REST dargestellt werden kann der Architekt zwischen alle am Markt üblichen Technologien wählen. Wichtig dabei ist allerdings, dass er ein Format wählt welches möglichst viele Geräte unterstützen.

3 REST Constrains

Die REST Architektur kann in Sechs wesentlich "Constrains"(Bedingungen) unterteilt werden, die unabhängig voneinander erfüllt werden. Diese stellen die unterschiedlichen Qualitätseigenschaften die REST bietet dar. [1]
Dabei lässt sich sagen, dass im wesentlich drauf gesetzt wird eindeutige Identifikatoren zu finden, Verknüpfungen(Hypermedia) zwischen diesen zu schaffen und den Zugriff durch standardisierte Methoden zu gewährleisten.

3.1 Client-Server

Jeder Anfrage wird durch den User (Client) initialisiert und wird komplett vom Server bearbeitet. Dabei achtet die REST Architekturstil auf eine strikte Trennung von Client und Server. Dies hat zur folge, dass sich eine klare Trennung der Verantwortlichkeiten ergibt. Dadurch ist auch eine sperate Entwicklung von Client und Server möglich. Dies erhöht die "maintainability"(Die Wahrscheinlichkeit nach einem Error in einer bestimmten Zeit wieder den normal Zustand herstellen zu können). [1] [3]

3.2 Statless

Der Stateless (Statuslos) Constraint verlangt, dass in jeder Kommunikation zwischen Client und Server alle Informationen zum Verständnis der Anfrage enthalten sein müssen. Das hat zur Folge das der Status der Konversation nicht auf dem Server sonder auf dem Client festgehalten wird. Dies ermöglicht die Server zu Duplizieren , was durch load balancing (Verteilung von Anfragen auf mehrere parallel Systeme) zur Erhöhung der Verfügbarkeit, Verlässlichkeit und der Skalierbarkeit führt. [1]

3.3 Cache

Der Cache wurde zum REST Architekturstiel hinzugefügt um durch das weglassen bestimmter Interaktionen die Effizienz und Skalierbarkeit zu erhöhen. Dies erhöht ebenfalls die Fehlertoleranz, da auch bei Internen Fehlern weiterhin die Inhalte an den Client gesendet werden können. Der Cache kann z.B durch das gern bei REST Anwendungen genutzte Hypertext Transfer Protocol (HTTP) realisiert werden. Cache kann aber auch in einer "intermediary tiers"(Vermittler Ebenen) auftauchen wenn es sich um ein vielschichtiges System handelt z.B bei einer Serviceorientierte Architektur (SOA). [1]

3.4 HATEOAS

HATEOAS steht für Hypermedia As The Engine Of Application State. Der Constraint verlangt, dass der Client nur mit "hypermedia" die Dynamische vom Server bereit gestellt werden interagiert. Dabei muss "hypermedia" nicht nur ein

reiner Link sein, z.B auch HTML Formulare fallen unter diese Kategorie. Dies hat zu folge das der Client kein Verständnis dafür braucht wie er mit jedem Service zu interagieren hat. So hebt REST stark von anderen Architekturstielen ab. Dies ist vor allem in HumanWeb verbreitet da hier der Client die möglichkeit besitzt Namen zu Interpretieren. [1]

4 Ressourcen Design

Eine Ressource ist die Schnittstelle zwischen Client und Server. Es ist somit entscheident für die Systemarchitektur wie die Ressourcen designt sind Dabei gibt es bestimmte Kategorien in die man eine Ressource einteilen kann. Die meistens der Ressourcen einer REST Anwendung sind sogenannte Primärressourcen bzw. Subressourcen. Diese bilden die fachlichen Kernkompetenzen der des Systems ab. Sie sind meist die persistenten Entitäten des Systems. Um diese Ressourcen bzw. Entitäten gut zu verwalten werden oft Listenressourcen benutzt die eine Aufzählung von Primär und Subressourcen darstellen. Falls diese Ressourcen allerdings zu ausführlich sind bzw. falls Untergruppen abgebildet werden sollen, kommen Filterressorucen zum Einsatz. Sie enthalten alle Ressourcen einer Listenressource die ein bestimmtes Filterkriterium erfüllen. In änlicher weise Funktionieren auch Paginierungsressourcen. Diese bilden Allerdings nur bestimmte Anzahl von Ressourcen noch einer festen Ordnung ab. Oft werden auch zum Abbilden von Prozessen sogenannte Aktivitäsressourcen verwendet um interne Abläuft sinnvoll abzubilden bzw. einzelne Verarbeitungsschritte zu speichern. Um das Verständnis der Beziehungen der verwendeten Ressourcen noch besser abzubilden wurde Konzeptressourcen eingeführt diese fassen auf einer Metaebene Beziehungen zwischen Ressourcen zusammen. Hierbei wurde sich allerdings nicht auf eine bestimmte Repräsentation geeinigt. Konzeptressourcen können auch nur zum abbilden von Verweisen auf Informationsressourcen genutzt werden. Allgemein wird nicht Definiert welche Typen Ressourcen in einer REST Architektur verwendet werden muss. Es ist auch erlaubt die selbe Ressource unter verschiedenen URI's anzubieten.

5 Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) ist für die Übertragung von Daten in Rechnernetzen gedacht. Es ist Zustandslos da in jeder Kommunikation alle Informationen zum bearbeiten der Anfrage enthalten sind. Im folgendem Abschnitt werden erst alle Methoden die Http unterstützt beschrieben. Dabei wird auch auf verschieden Caching Architekturen eingegangen. Danach folgt ein kurzer abschnitt zur Erstellung neuer Methoden in HTTP und zur derzeit noch nicht im Standard vorhandenen Methoden. [2] [4] Um die Darstellung von HTTP abzurunden wird ebenfalls auf Authentifizierung und Statuscodes in HTTP eingegangen.

5.1 GET-Methode

Die GET Methode wird zur Abfrage von Repräsentation einer Ressource genutzt. Sie ist dabei nicht gedacht Informationen in das System einzupflegen. Bei einer GET-Methode, in Http können an die URI der Ressource noch Übergabeparameter angehängt. Dies gehschieht nach folgenden Schema:
HTTP:Hierarchische-Struktur?[{Variablenname} = {Wert}&].
Dabei ist Prinzipiell egal wie viele Variablen dem Service übergeben werden. Bei Anfrage kann ebenfalls ein Format für die Antwort angegeben werden. In der Abbildung erwartet der Client z.B. ein JSON Objekt.

```
GET /orders HTTP/1.1
User-Agent: curl/7.37.1
Host: om.example.com
Accept: application/json
```

Fig. 1. Quelle: REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web (3. Auflage, 2015)

5.2 PUT-Methode

Bei einer PUT-Methode wird eine noch nicht vorhandene Ressource erzeugt bzw. verändert falls diese schon vorhanden sein sollte. Der Client spezifiziert hierzu im Http-Head der Nachricht das Format indem die Informationen über die Ressource übermittelt werden. Hierbei wird mit PUT meist eine ganze Ressource definiert. Dadurch kommt es zu keinen Unverständlichkeiten bei Updates, somit ist die Operation wie auch GET, HEAD, DELETE Idempotent. Dies bedeutet, dass es den Zustand des Systems nicht verändert ob die Anfrage ein oder mehrmals ausgeführt wird. PUT garantiert dem Client nur das die Ressource sinngemäß der gegebenen Information vom Server angelegt wird. Dabei ist es möglich das der Server Informationen des Clients weglässt.

5.3 POST-Methode

POST wird laut Definition dazu benutzt eine neue Ressource anzulegen. Dabei wird vom Client nur die Information über die Ressource an den Server weiter gegeben. Hierbei entscheidet der Server selbst unter welcher URI er die Ressource anlegt. Hiermit ist auch der größte Nachteil der Post-Methode verbunden, sie ist nicht Idempotent. Die Methode ist trotzdem hilfreich wenn z.B eine neue Ressource angelegt wird, die Teil einer Listen Ressource ist, so kann der Server nicht nur die URI der Ressource festlegen sondern diese auch der Listen Ressource hinzufügen.
Der Client wird durch den Statuscode 201 ("Created") vom Server informiert, dass die Ressource angelegt wurde. Darüber hinaus kann POST auch verwendet werden um jegliche Art von Interaktionskette anzustoßen, falls sich keine andere geeignete Http-Methode für die Funktionalität findet. Dadurch wird POST oft als Tunnel Methode missbraucht.

5.4 DELETE-Methode

Diese Methode ist, wie oben bereits erwähnt, auch Idempotent. Es ergibt sich keine Zustandsänderung egal ob das Löschen der in der Methode spezifizierten Ressource mehrmals oder nur einmal ausgeführt wird. Das hier definierte Löschen einer Ressource führ allerdings meistens nur dazu, dass die Ressource von außerhalb nicht mehr gesehen und adressiert werden kann. Sie kann jedoch intern im System noch vorhanden sein und zu einem späteren Zeitpunkt wieder sichtbar (referenzierbar) gemacht werden.

5.5 OPTION-Methode

Die Methode OPTION liefert Metadaten zu der adressierten Ressource zurück. Dabei wird z.B angegeben welche Methoden die Ressource unterstüzt. Die Implementierung dieser Methode ist nicht zwingend notwendig sollte allerdings vorgenommen werden.

5.6 CONNECT-Methode

Diese Methode wird zum Tunneln von anfragen über Proxys oder SSL genutzt. Dabei leitet die Zwischeninstanz die Anfrage des Clients ohne Veränderung an die Spezifizierte URI weiter.

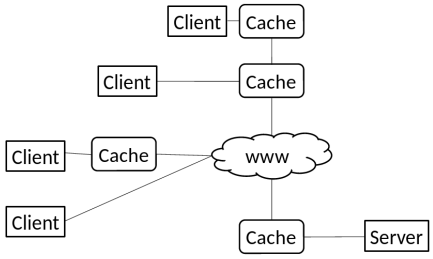
5.7 TRACE-Methode

Diese Methode ist oft bei realen Anwendung deaktiviert, aufgrund ihres Sicherheitsrisikos für das System. Wenn ein Server einen TRACE erhält, so soll er diesen genau so an den Client zurücksenden. Jeder Proxy der auf dem Weg zwischen Client und Server liegt ist dazu gezwungen einen Vermerk im Header der Nachricht über seine Existenz zu hinterlassen. Somit kann der Client genau den Weg seiner Anfragen nachverfolgen.

5.8 HTTP-Caching

Im folgenden werden die Mechanismen zur Cachkontrolle in HTTP erläutert, hierbei wird auch auf mögliche Architekturen von Cachingsystemen eingegangen. Beim Freshness/Expirationsmodell ist es dem Client möglich eine Antwort die er bereits erhalten hat wieder herzustellen. Diese Freshness kann sowohl durch den Server als auch durch den Client gesteuert werden. Dazu wird meist eine Zeitangabe an das Dokument angehängt, welche die Lebenszeit des Dokumentes angibt. Dazu wird im Header folgendes Attribut angegeben "Cache-Control: private, max-age=60". Hier steht die 60 für die Anzahl an Sekunden die das Dokument gültig ist.
Im Gegensatz dazu wird das Validation/Validierungsmodell verwendet um zu Überprüfen ob die gespeicherte Antwort noch aktuell ist. Dazu können Antworten einen Last-Modified-Header besitzen. Hierbei kann der Cache eine If-Modified-Since-Header Abfrage senden um die Validität des Dokuments zu überprüfen. Hierdurch entfällt in manchen fällen eine unnötige doppelte Übertragung der Informationen. Durch diesen Mechnanismus ist auch eine Invalidation möglich. Dies ist ein Mechanismus, der wenn eine zwischengespeicherte Antwort vorliegen, diese allerdings per POST/PUT/DELETE angesprochen wurde, jene Invalidiert. Es wird verhindert das eine Lost Update Problematik entsteht.

Architektur



In der der Abgebildeten Beispielarchitektur werden alle Arten von Caching dargestellt. Bei dem System handelt es sich um eine Webapplikation.

1. Client-Caching: Hierbei legt jeder Client seinen eigenen Cach mit Dokumenten an. Somit kann es dazu kommen das verschiedene Clients verschiedene Versionen eines Dokuments haben können. Diese Art von Caching ist standardmäßig auf fast alle HTTP Clients implementiert. Client-Caching ist nützlich, wenn Clients oft auf die selbe Ressource zugreifen, diese Ressource allerdings bei den einzelnen Clients verschieden ist.
2. Client-Shared-Caching: Beim Shared Caching teilen sich mehrere Clients einen Cach. Es können nicht Personalisierte Ressourcen gecacht werden. Dies ist meist sinnvoll wenn alle Clients auf die selbe Ressource zugreifen möchten. Dabei entscheidet der Shared-Cach wann er den Server für eine neue Repräsentation der Ressource anfragt.

3. Server-Cach: Es ist ebenfalls möglich alle Anfragen an den Server durch einen Serverseitigen-Cach zu leiten. Dabei werden alle anfragen die gecached auf dem Zwischenserver liegen, von ihm beantwortet. Nur Anfragen von Ressourcen die noch nicht gecached sind werden an den Server weitergeleitet.

[3] Die drei oben genannten Caching Architekturen können alle gleichzeitig in einem System genutzt werden, um das Optimum an Performance zu erreichen. Dabei muss sich bei der korrekten Verwendung von HTTP Headern meist nicht um die Verwaltung des Caches gekümmert werden.

5.9 Eigene Http-Methoden

Prinzipiell ist es möglich sich eigene Http Verben zu Definieren und diese als Methoden zu verwenden. Dies bringt allerdings einiges an Nachteilen mit sich und sollte nur in Ausnahmefällen angewendet werden. Die größten Nachteile hierbei sind die Sicherheite bzw. Idempotenz die bei den meisten eigenen Operationen nicht gewährleistet sind. Des weiteren kennt nur das abgeschlossene System die Bedeutung der Methode und auch nur wenn diese auf jedem Gerät ausreichend spezifiziert ist. Damit gehen fast alle Vorteile die eine REST Architektur bietet verloren da sie nicht mehr universell und stateless ist. Dennoch gibt es einige Projekte wie z.B. WebDAV(Web-based Distributed Authoring and Versioning) welches zusätzliche Methoden wie LOCK oder UNLOCK definiert um eine Dateiverwaltung über Http besser nutzbar zu machen.

5.10 Http-Authentifizierung

Seit der Version 1.0 besitzt Http eine Authentifizierungs-Methode. Diese nennt sich "Basic Authentication" und zwingt den Client bei den angegeben Ressourcen immer sein Benutzername und sein Passwort getrennt durch einen Doppelpunkt in Base64 zu übermitteln. Da Base64 nur ein Kodierungsstandart und verwendet wird um einen einheitlichen Zeichensatz zu erzeugen und keine Verschlüsselung darstellt ist diese Art der Authentifizierung sehr anfällig gegen Angriffe wie z.B. eine "Man-in-the-Middle"-Attacke. Deshalb wird häufig SSL bzw. HTTPS genutzt um die gesendeten Information zwischen Client und Server zu verschlüsseln. Hierbei sei erwähnt das die üblich verwendete Technick von Authentifizierung per Cookie zwar statless ist aber doch einige schwierigkeiten für die Sicherheit von REST Architekturen mit sich bringt.

Digest Authentication Seit der Http Version 1.1 gibt es der Basic Authentication eine weiter Methode der Authentifizierung die "Man-in-the-Middle"-Attacke erschweren soll. Hierbei wird die Kombination aus Benutzername und Passwort (Optional auch einem Salt) gehascht. Dies geschieht entweder durch das, als gebrochen geltende, md5 Haschisch oder durch SHA1. Dies verhindert bei einem unverschlüsselten übertragen zwar, dass der Angreifer das Passwort bekommt, es ist allerdings dennoch möglich die Anfrage an den Server zu verändern ohne dass der Client oder Server davon etwas mitbekommt.

Realität In der Realität bietet die "Http-Authentication" oft nicht die gewünschten Funktionen. So kann es gewünscht sein auf jeder Seite einen Loginscreen einzublenden, dass der Nutzer sich jeder Zeit mit seinem Profil verbinden kann. Des weiteren werden oft weitere Elemente bei einem Login benötigt als nur ein Feld für Benutzernamen und Passwort. Dies ist allerdings in Http nicht vorgesehen. Aus diesen Gründen wir oft eine Session basierte Architektur eingesetzt, welche mit Cookies arbeitet. [3]

5.11 HTTP-Statuscodes

Um dem Client Information darüber zu liefern wie seine Anfrage verarbeitet wurde gibt es in HTTP Statuscodes die in verschiedene Kategorien unterteilt sind. Im folgende werden alle Kategorien tabellarisch aufgelistet. Anschließend wird noch auf wichtige Statuscodes näher eingegangen .

Kategorie	Beschreibung
1XX	Informationen (gibt dem Client Feedback, dass die Bearbeitung seiner Anfrage noch andauert)
2XX	Erfolgreich (gibt dem Client Feedback, dass seine Anfrage erfolgreich war)
3XX	Umleitung (Um die Anfrage umzusetzen benötigt der Server weitere Informationen vom Client)
4XX	Client-Fehler (Der Client hat vermutlich eine nicht korrekte Anfrage gestellt)
5XX	Server-Fehler (Der Server kann vermutlich die Anfrage nicht Korrekt Verarbeiten)
9XX	Eigene Fehlermeldungen

Wichtigste Statuscodes Hier noch eine Liste der am Wichtigsten und Häufigst benutzten Codes:

Statuscode	Beschreibung
200	Die Anfrage wurde bearbeitet und Antwort wird mit dem Code übertragen
201	Die angeforderte Ressource wurde erstellt
400	Die Anfrage war fehlerhaft aufgebaut
403	Die Anfrage wird aufgrund von fehlenden Berechtigungen des Clients nicht ausgeführt
404	Die angefragte Ressource wurde nicht gefunden
405	Die angefragte Methode ist für die Ressource nicht erlaubt
500	Interner Fehler im Server
502	Der Server konnte nicht als "Gateway" bzw. Proxy fungieren

6 REST Repräsentationsformate

Dieses Kapitel befasst sich in erster Linie mit den gängigen Datenaustauschformaten. Es werden auch die Vorteile der Formaten diskutiert, da REST die Möglichkeit bietet hinter einer Ressource mehrere Repräsentationen zu hinterlegen.

6.1 XML

Die Extensible Markup Language ist eine, vom W3C herausgegebene Auszeichnungssprache. Sie baut sich durch Elemente auf, die in einer Baumstruktur angeordnet werden können. Um die Validität eines Elementes zu überprüfen kann ein optionales XML Schema definiert werden. Bei XML ist es sommit möglich eigene Entitätsstrukturen zu entwickeln oder bereits vorhandene Strukturen zu verwenden um eine größere Zahl von Clients die Interpretation der Ressource zu erleichtern. Darüber hinaus enthält der XML Standart Atrribute, welche ein Element näher spezifizieren und oft zur Angabe von URI's genutzt werden. Da XML in der REST Welt weit verbeitet ist gibt es von der W3C auch viele Syntax Erweiterung (XLink) und Tools (XPath) die, die Verwendung in REST erleichtern indem sie die Hypermedia Verarbeitung standardisieren.

6.2 JSON

Die JavaScript Object Notation kurz JSON kommt wie der Name schon sagst ursprünglich aus der Scriptsprache Java Script. Dabei ist das Ziel von JSON eine Datenstruktur möglichst kompakt und gut lesbar darzustellen. Troz seiner großen Verbreitung biete es für die REST einige Probleme. So gibt es zwar JSON als spezifizierbaren Typ in Http ("application/json") , allerdings gibt es innerhalb von JSON keine festen Typisierung. Um diese Schwachstelle auszumerzen haben sich JSON bassierte Formate entwickelt wie z.B HAL (Hypermedia Application Language), welches durch hinzufügen von Attributen wie "_links" versucht JSON um Hypermedia Elemente zu erweitern.

6.3 CSV

Für das Comma Separated Value Format gibt es wie für die beiden anderen Foramte auch einen MIME-Type(Internet Media Type). Es wird unter dem Medientyp "text/csv" angegeben. Hier kann zu dem eigentlichen CSV noch die Zeichenkodierung angegeben werden. Es ist darüber hinaus auch möglich in dem optionalen Header anzugeben ob die Datei eine Kopfzeile enthält. Für manche Clients ist allerdings CSV nicht geeignet da sie durch anderen Zeichensätze Kommas falsch interpretieren. Allerdings ist gerade im Bereich von Listen Ressourcen die Repräsentation leichtgewichtig und einfach zu verstehen.

6.4 RDF

Das Resource Description Framework ist ein Interessanter versuch des W3C informationen anhand eines Graphen zu speicher. Dabei besteht eine RDF datei aus einer Menge von Tripel, die sich änlich wie natürliche Sprachen aufbauen. In dem Tripel wird für jede Teilmenge ein Subjekt ein Prädikat und ein Objekt Spezifiziert woraus sich ein Graph mit zwei Knoten und einer Kante(Relation) aufbauen lässt. Um allerdings hiermit RESTkonforme Daten abzubilden wird jedes Element als URI Abgebildet, wodurch sich jede Beziehung von Ressourcen modelieren lässt. Hierbei sei Erwähnt, dass RDF nur eine Framework ist und mit jeder der Repräsentationen implementiert werden kann.

References

1. Bruno Costa, Paulo F. Pires, Flavia C. Delicato, and Paulo Merson. Evaluating a Representational State Transfer (REST) architecture: What is the impact of REST in my architecture? *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*, pages 105–114, 2014.
2. Roy Fielding, James Gettys, Jeff Mogul, Henrik Frystyk, Larry Masinter, P Leach, and Tim Berners-Lee. RFC2616 - Hypertext transfer protocol–HTTP/1.1. *Internet Engineering Task Force*, pages 1–114, 1999.
3. Silvia Schreier Oliver Wolf Stefan Tilkov, Martin Eigenbrodt. *REST und HTTP Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt.verlag GmbH, Heidelberg, 2015.
4. Representational State Transfer. Resource-Oriented Architecture. pages 359–415.
5. http://www.w3schools.com/ (Stand 31.01.2017)
6. https://www.w3.org/ (Stand 15.02.2017)